Procesos. Problemas propuestos

Ejercicio 1

Suponga que el fragmento de código que aparece a continuación corresponde al esquema de operación de un servidor concurrente que debe estar en funcionamiento continuamente en un sistema de una determinada organización.

```
int main(...){
      struct peticion pet;
      pid_t pid;
      while(1){
            esperar_peticion(&pet);
            pid = fork();
            switch(pid){
            case -1:
                  perror("fork"); exit(1);
            case 0:
                  procesar_peticion(&pet); // operación muy larga
                  exit(0);
            }
      }
      return 0;
}
```

a) ¿Qué problema de agotamiento de recursos puede presentarse al usar este esquema?

Para paliar este problema sin afectar al paralelismo de la solución, uno de los programadores más expertos de la organización propone este esquema alternativo.

```
int main(...){
      struct peticion pet;
      pid_t pid1, pid2;
      while(1){
            esperar_peticion(&pet);
            pid1 = fork();
            switch(pid1){
            case -1:
                  perror("fork"); exit(1);
            case 0:
                  if ((pid2 = fork()) == -1) {
                        perror("fork");
                        exit(1);
                  }
```

```
                else if (pid2 > 0)
                        exit(0);
                else {
                        procesar_peticion(&pet); // operación muy larga
                        exit(0);
                }
        default:
                wait(NULL);
        }
    }
    return 0;
}
```

b) Explique cómo resuelve este nuevo esquema el problema identificado.

Ejercicio 2

Dado el siguiente programa:

a) ¿Qué contiene la variable p1?

b) ¿Qué sentencias del cuerpo del switch corresponden al proceso hijo?

c) ¿Qué función realiza el proceso hijo? (Recuerde que “a % b” es el resto de la división entera de a entre b)

d) ¿Qué función realiza la macro WEXITSTATUS()?

e) ¿Qué operaciones hace el proceso padre tras crear al proceso hijo?

f) ¿Cuántos procesos hijo pueden existir al mismo tiempo? ¿Por qué?

g)¿Qué información contienen las variables k y n al final de cada iteración del bucle for?

h) ¿Puede en algún caso caso iterar indefinidamente el bucle for? Si es así, ¿en cuál?

i) ¿Qué información contiene la variable k al final del programa?

j) ¿Qué escribe el programa en la salida estándar si el sistema operativo asigna la siguiente secuencia de PIDs a los procesos hijo: 29180, 29181, 29182, 29183, 29184, 29185, 29186, 29187, etc.?

PROGRAMA:

```
int main(void){
    int s, n, k = 0;
    pid_t p1;
    for(n = 0; n < 3; ){
        p1 = fork();
        switch(p1){
        case -1:
            perror("fork"); exit(1);
        case 0:
```

```
                  if(getpid()%2 == 0)
                        exit(0);
                  else
                        exit(1);
            default:
                  k++;
                  wait(&s);
                  if(WIFEXITED(s))
                        if(WEXITSTATUS(s) == 0)
                              n++;
            }
      }
      printf("k = %i\n", k);
      return 0;
}
(FIN PROGRAMA)
```

Ejercicio 3

Escribir un programa que cree N generaciones padre-hijo. Cada pareja padre-hijo se deben comunicar con un pipe unidireccional padre->hijo.

Ejercicio 4

Escribir un programa que cree N procesos hijo, tomando N del primer argumento de la línea de mandato. Cada pareja de hijos consecutivos i e i+1 se deben comunicar entre sí con un pipe unidireccional hijo i --> hijo i+1. El último hijo debe comunicarse mediante pipe unidireccional con el primer hijo (último hijo → primer hijo)

Ejercicio 5

Escribir un programa que cree cuatro procesos, que denominaremos de forma lógica P0, P1, P2 y P3. Estos cuatro procesos estarán comunicados entre ellos a través de un único pipe, de solo lectura para P0 y solo escritura para P1, P2 y P3. Los procesos P1, P2 y P3 escribirán en el pipe 1 carácter que los identifica ('1' = P1, '2' = P2 y '3' = P3) y terminarán y el proceso P0 leerá los caracteres y los imprimirá por stdout.

Ejercicio 6

Parte 1:

Se pide implementar en C y para UNIX un programa con las siguientes características:

a) Se debe invocar de la siguiente manera:

verificar_actividad minutos mandato [args...]

b) Su objetivo es verificar, con cierta periodicidad, que un operario de una central nuclear está

alerta.

c) El programa debe alternar fases de espera de los minutos especificados como primer argumento en la invocación, con fases de verificación de presencia del operario.

d) Para la fase de verificación deberá crearse un proceso hijo cuyo estado de terminación indique si todo ha ido de forma correcta (terminación 0) o si ha habido algún error (terminación no 0).

e) En la fase de verificación se debe presentar un mensaje como “Introduzca 4:”, siendo el dígito mostrado el menos significativo del valor del pid del proceso hijo. El operario dispondrá de 30 segundos para introducir correctamente el dígito presentado. Durante este tiempo, cada 2 segundos se llamará la atención del operario con una señal acústica producida por la función void beep (void) de la biblioteca curses.

f) Si la segunda fase va mal el proceso principal deberá terminar ejecutando el mandato (con sus respectivos argumentos) indicado en la invocación del programa.

g) Tanto en el proceso padre como en los procesos hijos creados se deberán enmascarar las señales generables desde el teclado INT y QUIT.

RECUERDE QUE: Cuando llega una señal a un proceso que está bloqueado en una llamada al sistema, la llamada se aborta, devolviendo un -1, y la variable errno toma el valor EINTR.

Parte 2:

Escribir un programa complementario al anterior que cree un proceso que se encargue de comprobar periódicamente que el proceso verificador existe, y, en caso contrario, que lo ponga en ejecución.

Ejercicio 7

Escribir un programa que cree dos procesos que actúen de productor y de consumidor respectivamente. El productor abrirá el fichero denominado /tmp/datos.txt y leerá los caracteres almacenados en él. El consumidor tendrá que calcular e imprimir por la salida estándar el número de caracteres almacenados en ese fichero sin leer del fichero. La comunicación entre los dos procesos debe hacerse utilizando un pipe. Por otra parte, el proceso padre tendrá que abortar la ejecución de los dos procesos hijos, si transcurridos 60 segundos éstos no han acabado su ejecución.

Ejercicio 8

Escribir un programa que cree N threads joinables. La función principal main() se debe comunicar con los threads mediante un vector de N structs (con diversos campos: campo1, campo2, campo3, etc). Cada elemento i del vector sirve para comunicar a main() con el thread i y no debe poderse acceder por el resto de threads.

Ejercicio 9

Se desea desarrollar una aplicación que debe realizar dos tareas que se pueden ejecutar de forma independiente. Los códigos de estas dos tareas se encuentran definidos en dos funciones cuyos prototipos en lenguaje de programación C, son los siguientes:

```
void tarea_A(void);

void tarea_B(void);
```

Se pide:

a) Programar la aplicación anterior utilizando tres modelos distintos:

1. Un único proceso.

2. Procesos para aprovechar paralelismo. Plantear una solución que minimice el número de procesos creados y solicite el mínimo número de servicios al sistema operativo.

3. Procesos ligeros para aprovechar paralelismo. En este caso también se minimizará el número de procesos ligeros creados y el número de servicios.

En cualquiera de los tres casos la aplicación debe terminar cuando se hayan acabado las tareas A y B.

Ejercicio 10

Desarrolle un programa que reciba como argumentos un tamaño en bytes y un conjunto de ficheros, y que imprima cuántas líneas hay en cada uno de los mismos. Esta operación se realizará de forma concurrente mediante hilos usando el siguiente esquema: por cada fichero se crearán tantos hilos como fragmentos del tamaño indicado como primer argumento haya en el fichero, redondeado por exceso, de manera que cada hilo se encargue del fragmento correspondiente (así, por ejemplo, si el tamaño especificado como primer argumento es 40.960 bytes y el tamaño del fichero es 200.000 bytes, se crearán 5 hilos, el último de los cuales se encargará de los últimos 36.160 bytes). Se propone diseñar una solución que no requiera ningún tipo de sincronización explícita entre los hilos, más allá de la que existe implícitamente en las operaciones de creación y terminación de los hilos.